CHAPTER

8

Learning System Administration

IN THIS CHAPTER

Doing graphical administration

Using the root login

Understanding administrative commands, config files, and log files

Working with devices and filesystems

Linux, like other UNIX-based systems, was intended for use by more than one person at a time. *Multiuser features* enable many people to have accounts on a single Linux system with their data kept secure from others. *Multitasking* enables many people to run many programs on the computer at the same time, with each person able to run more than one program. Sophisticated networking protocols and applications make it possible for a Linux system to extend its capabilities to network users and computers around the world. The person assigned to manage all of a Linux system's resources is called the *system administrator*.

Even if you are the only person using a Linux system, system administration is still set up to be separate from other computer use. To do most administrative tasks, you need to be logged in as the *root user* (also called the *superuser*) or to get root permission temporarily (usually using the `sudo` command). Regular users who don't have root permission cannot change, or in some cases cannot even see, some of the configuration information for a Linux system. In particular, security features such as stored passwords are protected from general view.

Because Linux system administration is such a huge topic, this chapter focuses on the general principles of Linux system administration. In particular, it examines some of the basic tools that you need to administer a Linux system for a personal desktop or on a small server. Beyond the basics, this chapter also teaches you how to work with filesystems and monitor the setup and performance of your Linux system.

Understanding System Administration

Separating the role of system administrator from that of other users has several effects. For a system that has many people using it, limiting who can manage it enables you to keep it more

Linux® Bible, Tenth Edition. Christopher Negus.

secure. A separate administrative role also prevents others from casually harming your system when they are just using it to write a document or browse the Internet.

If you are the system administrator of a Linux system, you generally log in as a regular user account and then ask for administrative privileges when you need them. This is often done with one of the following:

`su` command: Often, `su` is used to open a shell as root user. After the shell is open, the administrator can run multiple commands and then exit to return to a shell as a regular user.

`sudo` command: With `sudo`, a regular user is given root privileges, but only when that user runs the `sudo` command to run another command. After running that one command with `sudo`, the user is immediately returned to a shell and acts as the regular user again. Ubuntu and Fedora by default assign `sudo` privilege to the first user account when those systems are installed. This is not done by default in RHEL, although during RHEL installation, you can choose for your first user to have `sudo` privilege if you'd like.

Cockpit browser-based administration: RHEL, Fedora, and other Linux distributions have committed to Cockpit as their primary browser-based system administration facility. With Cockpit enabled, you can monitor and change your system's general activities, storage, networking, accounts, services, and other features.

Graphical windows: Before Cockpit was widely available, RHEL, Fedora, and other Linux distributions offered individual graphical administration tools that were launched by commands beginning with `system-config-*`. Although most of these administration tools are not being offered in the latest release of RHEL and Fedora, they are noted here because they are still available in older Linux releases.

Tasks that can be done only by the root user tend to be those that affect the system as a whole or impact the security or health of the system. Following is a list of common features that a system administrator is expected to manage:

Filesystems: When you first install Linux, the directory structure is set up to make the system usable. However, if users later want to add extra storage or change the filesystem layout outside of their home directory, they need administrative privileges to do that. Also, the root user has permission to access files owned by any user. As a result, the root user can copy, move, or change any other user's files—a privilege needed to make backup copies of the filesystem for safekeeping.

Software installation: Because malicious software can harm your system or make it insecure, you need root privilege to install software so that it is available to all users on your system. Regular users can still install some software in their own directories and can list information about installed system software.

User accounts: Only the root user can add and remove user accounts and group accounts.

Network interfaces: In the past, the root user had to configure network interfaces and start and stop those interfaces. Now, many Linux desktops allow regular users to start and stop network interfaces from their desktop using Network Manager. This is particularly true for wireless network interfaces, which can come and go by location as you move your Linux laptop or handheld device around.

Servers: Configuring web servers, file servers, domain name servers, mail servers, and dozens of other servers requires root privilege, as does starting and stopping those services. Content, such as web pages, can be added to servers by non-root users if you configure your system to allow that. Services are often run as special administrative user accounts, such as `apache` (for the `httpd` service) and `rpc` (for the `rpcbind` service). So, if someone cracks a service, they can't get root privilege to other services or system resources.

Security features: Setting up security features, such as firewalls and user access lists, is usually done with root privilege. It's also up to the root user to monitor how the services are being used and to make sure that server resources are not exhausted or abused.

The easiest way to begin system administration is to use some graphical administration tools.

Using Graphical Administration Tools

Most system administration for the first Linux systems was done from the command line. As Linux became more popular, however, both graphical and command-line interfaces began to be offered for most common Linux administrative tasks.

The following sections describe some of the point-and-click types of interfaces that are available for doing system administration in Linux.

Using Cockpit browser-based administration

Cockpit is the best browser-based Linux system administration tool that I have ever seen. It brings together a range of Linux administrative activities into one interface and taps into a diverse set of Linux APIs using cockpit-bridge. As someone doing Linux administration, however, you just need to know that you will get a consistent and stable way of administering your systems with Cockpit.

Getting started with Cockpit is as simple as enabling the cockpit socket and pointing a web browser at the Cockpit service. Because of Cockpit's plug-in design, there are new tools being created all the time that you can add to your system's Cockpit interface.

If you are starting with the latest RHEL or Fedora systems, performing the following procedure lets you enable and start using Cockpit on your system.

Note

No configuration is required to start this procedure. However, you can configure Cockpit to use your own OpenSSL certificate instead of the self-signed one used by default. This lets you avoid having to accept the unverified self-signed certificate when you open the Cockpit interface from your browser.

1. If Cockpit is not already installed, do the following:

```
# dnf install cockpit
```

2. Log in as root user, and enable the Cockpit socket:

```
# systemctl enable --now cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/
cockpit.socket
    → /usr/lib/systemd/system/cockpit.socket.
```

3. Open your web browser to port 9090 on the system where you just enabled Cockpit. You can use the hostname or IP address. Port 9090 is configured for https by default, although you can reconfigure that if you like to use http. Here are examples of addresses to type into your browser's address bar:

```
https://host1.example.com:9090/
https://192.168.122.114:9090/
```

4. Assuming you didn't replace the self-signed certificate for Cockpit, you are warned that the connection is not safe. To accept it anyway, and depending on your browser, you must select Advanced and agree to an exception to allow the browser to use the Cockpit service.
5. Enter your username and password. Use the root user or a user with `sudo` privileges if you want to change your system configuration. A regular user can see but not change most settings. Figure 8.1 shows an example of this window.
6. Begin using Cockpit. The Cockpit dashboard contains a good set of features by default (you can add more later) on RHEL and Fedora systems. Figure 8.2 shows an example of the System area of the Cockpit dashboard:

Immediately after logging in to Cockpit, you begin seeing system activity related to CPU usage, memory and swap, disk input/output, and network traffic. Selections in the left navigation pane let you begin working with logs, storage, networking, user and group accounts, services, and many other features on your system.

As you proceed through the rest of this book, you will see descriptions of how to use the different features of Cockpit in the appropriate section. To dive deeper into any of the topics that you encounter with Cockpit, I recommend checking out the Cockpit project website: `https://cockpit-project.org`.

FIGURE 8.1

Logging in to Cockpit

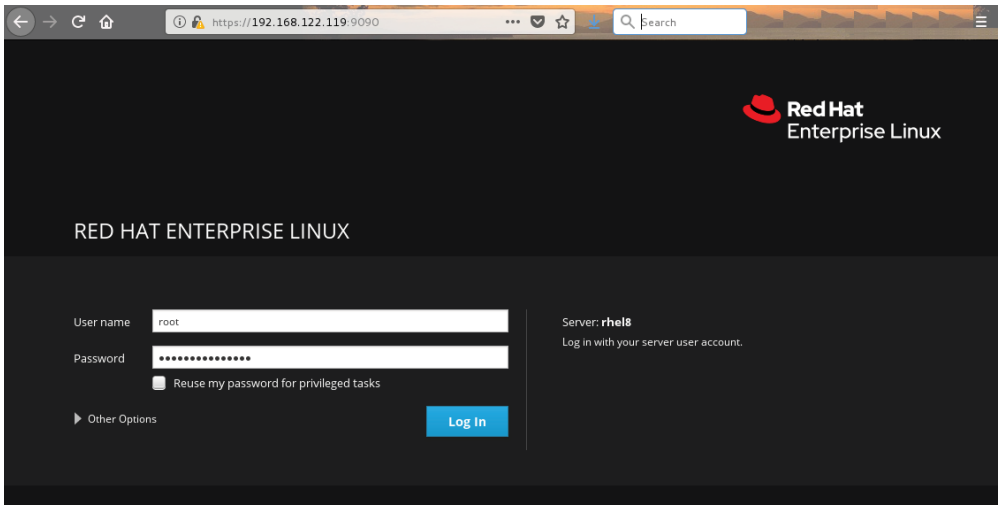

FIGURE 8.2

View system activity and other topics from the Cockpit dashboard.

Using system-config-* tools

On Fedora and RHEL systems prior to the release of Cockpit, a set of graphical tools was available from the Administration submenu of the System menu (GNOME 2), from the

Activities screen (GNOME 3), or from the command line. On these older Fedora and RHEL systems, you could operate these tools from the command line by running a set of commands that began with the `system-config*` string (such as `system-config-network`).

These `system-config*` tools require root permission. If you are logged in as a regular user, you must enter the root password before the graphical user interface (GUI) application's window opens or, in some cases, when you request to do some special activity.

The following list describes many of the graphical tools available in earlier Fedora or RHEL systems. (Some were only in Fedora and many are not installed by default.) The command that you would launch to get the feature is shown in parentheses (often, it is the same as the package name). The following graphical tools were available in Fedora:

Domain Name System (`system-config-bind`): Create and configure zones if your computer is acting as a DNS server.

HTTP (`system-config-httpd`): Configure your computer as an Apache web server.

NFS (`system-config-nfs`): Set up directories from your system to be shared with other computers on your network using the NFS service.

Root Password (`system-config-rootpassword`): Change the root password.

Samba NFS (`system-config-samba`): Configure Windows (SMB) file sharing. (To configure other Samba features, you can use the SWAT window.)

The following graphical tools were available in both Fedora and RHEL systems prior to RHEL 8:

Services (`system-config-services`): Display and change which services are running on your Fedora system at different runlevels from the Service Configuration window.

Authentication (`system-config-authentication`): Change how users are authenticated on your system. Typically, shadow passwords and MD5 passwords are selected. However, if your network supports LDAP, Kerberos, SMB, NIS, or Hesiod authentication, you can select to use any of those authentication types.

Date & Time (`system-config-date`): Set the date and time or choose to have an NTP server keep system time in sync.

Firewall (`system-config-firewall`): Configure your firewall to allow or deny services to computers from the network.

Language (`system-config-language`): Select the default language used for the system.

Printing (`system-config-printer`): Configure local and network printers.

SELinux Management (`system-config-selinux`): Set SELinux enforcing modes and default policy.

Users & Groups (`system-config-users`): Add, display, and change user and group accounts for your Fedora system.

Other administrative utilities were available from the Applications menu on the top panel. Select the System Tools submenu (in GNOME 2) or go to the Activities screen (in GNOME 3) to choose some of the following tools (if installed):

Configuration Editor (`gconf-editor`): Directly edit the GNOME configuration database.

Disk Usage Analyzer (`gnome-utils`): Display detailed information about your hard disks and removable storage devices.

Disk Utility (`gnome-disks`): Manage disk partitions and add filesystems (`gnome-disk-utility` package).

Kickstart (`system-config-kickstart`): Create a kickstart configuration file that can be used to install multiple Linux systems without user interaction.

Descriptions from previous editions of this book of most of these tools have been replaced by procedures using Cockpit instead.

Using other browser-based admin tools

To simplify the management of many enterprise-quality open source projects, those projects have begun offering browser-based graphical management tools. In most cases, command-line tools are offered for managing these projects as well.

For example, if you are using Red Hat Enterprise Linux, there are browser-based interfaces for managing the following projects:

Red Hat OpenShift: *OpenShift*, based on the Kubernetes project, offers a browser-based interface for deploying and managing a cluster of control plane and worker nodes as well as features for deploying and managing containers in what are referred to as *pods*. See the Red Hat OpenShift site at `www.openshift.com` or the upstream OKD site at `www.okd.io` for details.

Red Hat Enterprise Linux OpenStack Platform (RHELOSP): The OpenStack platform-as-a-service project lets you manage your own private, hybrid cloud through your browser. This includes the OpenStack dashboard from the OpenStack Horizon project (`http://horizondocs.openstack.org/horizon/latest`). That interface lets you launch and manage virtual machines and all of the resources around them: storage, networking, authentication, processing allocations, and so on. Refer to Chapter 27, "Using Linux for Cloud Computing," for a description of how to use the OpenStack Dashboard.

Red Hat Virtualization (RHV): With RHEV, the RHV manager provides the browser-based interface for managing virtual machines, including allocating storage and user access to resources. Many other examples of browser-based graphical administration tools are available with open source projects. If you are new to Linux, it can be easier to get started with these interfaces. However, keep in mind that often you need to use command-line tools if you need to troubleshoot problems because graphical tools are often limited in that area.

Using the root User Account

Every Linux system starts out with at least one administrative user account (the root user) and possibly one or more regular user accounts (given a name that you choose, or a name assigned by your Linux distribution). In most cases, you log in as a regular user and become the root user to do an administrative task.

The root user has complete control of the operation of your Linux system. That user can open any file or run any program. The root user also installs software packages and adds accounts for other people who use the system.

Tip

Think of the root user in Linux as similar to the Administrator user in Windows.

When you first install most Linux systems (although not all systems), you add a password for the root user. You must remember and protect this password; you need it to log in as root or to obtain root permission while you are logged in as some other user.

To become familiar with the root user account, you can simply log in as the root user. I recommend trying this from a virtual console. To do so, press Ctrl+Alt+F3. When you see the login prompt, type **`root`** (press Enter) and enter the password. A login session for root opens. When you are finished, type **`exit`**, and then press Ctrl+Alt+F1 to return to the regular desktop login.

After you have logged in as root, the home directory for the root user is typically `/root`. The home directory and other information associated with the root user account are located in the `/etc/passwd` file. Here's what the root entry looks like in the `/etc/passwd` file:

```
root:x:0:0:root:/root:/bin/bash
```

This shows that for the user named `root`, the user ID is set to `0` (root user), the group ID is set to `0` (root group), the home directory is `/root`, and the shell for that user is `/bin/bash`. (Linux uses the `/etc/shadow` file to store encrypted password data, so the password field here contains an `x`.) You can change the home directory or the shell used by editing the values in this file. A better way to change these values, however, is to use the `usermod` command (see the section "Modifying Users with usermod" in Chapter 11 for further information).

At this point, any command that you run from your shell is run with root privilege. So be careful. You have much more power to change (and damage) the system than you did as a regular user. Again, type **`exit`** when you are finished. If you are on a virtual console and have a desktop interface running on another console, press Ctrl+Alt+F1 to return to the graphical login screen if you are using a Linux desktop system.

NOTE
By default, the root account has no password set in Ubuntu. This means that even though the account exists, you cannot log in using it or use `su` to become the root user. This adds an additional level of security to Ubuntu and requires you to use `sudo` before each command that you want to execute as the root user.

Becoming root from the shell (su command)

Although you can become the superuser by logging in as root, sometimes that is not convenient.

For example, you may be logged in to a regular user account and just want to make a quick administrative change to your system without having to log out and log back in. You may need to log in over the network to make a change to a Linux system but find that the system doesn't allow root users in from over the network (a common practice for secure Linux systems). One solution is to use the `su` command. From any Terminal window or shell, you can simply type the following:

```
$ su
Password: ******
#
```

When you are prompted, type the root user's password. The prompt for the regular user (`$`) changes to the superuser prompt (#). At this point, you have full permission to run any command and use any file on the system. However, one thing that the `su` command doesn't do when used this way is read in the root user's environment. As a result, you may type a command that you know is available and get the message `Command Not Found`. To fix this problem, use the `su` command with the dash (-) option instead like this:

```
$ su -
Password: ******
#
```

You still need to type the password, but after that everything that normally happens at login for the root user happens after the `su` command is completed. Your current directory will be root's home directory (probably `/root`), and things such as the root user's `PATH` variable are used. If you become the root user by just typing `su`, rather than `su -`, you don't change directories or the environment of the current login session.

You can also use the `su` command to become a user other than root. This is useful for troubleshooting a problem that is being experienced by a particular user but not by others on the computer (such as an inability to print or send email). For example, to have the permissions of a user named *jsmith*, you'd type the following:

```
$ su - jsmith
```

Even if you were root user before you typed this command, afterward you would have only the permissions to open files and run programs that are available to jsmith. As root user, however, after you type the `su` command to become another user, you don't need a password to continue. If you type that command as a regular user, you must type the new user's password.

When you are finished using superuser permissions, return to the previous shell by exiting the current shell. Do this by pressing Ctrl+D or by typing **`exit`**. If you are the administrator for a computer that is accessible to multiple users, don't leave a root shell open on someone else's screen unless you want to give that person freedom to do anything he or she wants to the computer!

Allowing administrative access via the GUI

As mentioned earlier, when you run GUI tools as a regular user (from Fedora, Red Hat Enterprise Linux, or some other Linux systems), you are prompted for the root password before you are able to access the tool. By entering the root password, you are given root privilege for that task.

For Linux systems using the GNOME 2 desktop, after you enter the password, a yellow badge icon appears in the top panel, indicating that root authorization is still available for other GUI tools to run from that desktop session. For GNOME 3 desktops, you must enter the root password each time you start any of the system-config tools.

Gaining administrative access with sudo

Particular users can also be given administrative permissions for particular tasks or any task by typing `sudo` followed by the command they want to run, without being given the root password. The `sudoers` facility is the most common way to provide such privilege. Using `sudoers` for any users or groups on the system, you can do the following:

- Assign root privilege for any command they run with `sudo`.
- Assign root privilege for a select set of commands.
- Give users root privilege without telling them the root password because they only have to provide their own user password to gain root privilege.
- Allow users, if you choose, to run `sudo` without entering a password at all.
- Track which users have run administrative commands on your system. (Using `su`, all you know is that someone with the root password logged in, whereas the `sudo` command logs which user runs an administrative command.)

With the `sudoers` facility, giving full or limited root privileges to any user simply entails adding the user to `/etc/sudoers` and defining what privilege you want that user to have. Then the user can run any command they are privileged to use by preceding that command with the `sudo` command.

Here's an example of how to use the `sudo` facility to cause the user named *joe* to have full root privilege.

Tip

If you look at the `sudoers` file in Ubuntu, you see that the initial user on the system already has privilege, by default, for the `sudo` group members. To give any other user the same privilege, you could simply add the additional user to the admin group when you run `visudo`.

1. As the root user, edit the `/etc/sudoers` file by running the `visudo` command:

   ```
   # /usr/sbin/visudo
   ```

   By default, the file opens in `vi`, unless your `EDITOR` variable happens to be set to some other editor acceptable to `visudo` (for example, `export EDITOR=gedit`). The reason for using `visudo` is that the command locks the `/etc/sudoers` file and does some basic sanity checking of the file to ensure that it has been edited correctly.

Note

If you are stuck here, try running the `vimtutor` command for a quick tutorial on using `vi` and `vim`.

2. Add the following line to allow joe to have full root privileges on the computer:

   ```
   joe     ALL=(ALL)     ALL
   ```

   This line causes joe to provide a password (his own password, not the root password) in order to use administrative commands. To allow joe to have that privilege without using a password, type the following line instead:

   ```
   joe     ALL=(ALL)     NOPASSWD: ALL
   ```

3. Save the changes to the `/etc/sudoers` file (in `vi`, type **Esc** and then **:wq**). The following is an example of a session by the user joe after he has been assigned `sudo` privileges:

   ```
   [joe]$ sudo touch /mnt/testfile.txt
     We trust you have received the usual lecture
     from the local System Administrator. It usually
     boils down to these two things:
       #1) Respect the privacy of others.
       #2) Think before you type.
   Password: *********
   [joe]$ ls -l /mnt/testfile.txt
   -rw-r--r--. 1 root root 0 Jan  7 08:42 /mnt/testfile.txt
   [joe]$ rm /mnt/testfile.txt
   rm: cannot remove `/mnt/testfile.txt': Permission denied
   [joe]$ sudo rm /mnt/textfile.txt
   [joe]$
   ```

In this session, the user joe runs the `sudo` command to create a file (`/mnt/textfile.txt`) in a directory for which he doesn't have write permission. He is given a warning and asked to provide his password (this is joe's password, *not* the root password).

Even after joe has entered the password, he must still use the `sudo` command to run subsequent administrative commands as root (the `rm` fails, but the `sudo rm` succeeds). Notice that he is not prompted for a password for the second `sudo`. That's because after entering his password successfully, he can enter as many `sudo` commands as he wants for the next five minutes, on RHEL and Fedora systems, without having to enter it again. For Ubuntu, this is set to zero, for no time-out. (You can change the time-out value from five minutes to any length of time you want by setting the `passwd_timeout` value in the `/etc/sudoers` file.)

The preceding example grants a simple all-or-nothing administrative privilege to joe. However, the `/etc/sudoers` file gives you an incredible amount of flexibility in permitting individual users and groups to use individual applications or groups of applications. Refer to the `sudoers` and `sudo` man pages for information about how to tune your `sudo` facility.

Exploring Administrative Commands, Configuration Files, and Log Files

You can expect to find many commands, configuration files, and log files in the same places in the filesystem, regardless of which Linux distribution you are using. The following sections give you some pointers on where to look for these important elements.

Note

If GUI administrative tools for Linux have become so good, why do you need to know about administrative files? For one thing, while GUI tools differ among Linux versions, many underlying configuration files are the same. So if you learn to work with them, you can work with almost any Linux system. Also, if a feature is broken, or if you need to do something that's not supported by the GUI, when you ask for help, Linux experts almost always tell you how to run commands or change the configuration file directly.

Administrative commands

Only the root user is intended to use many administrative commands. When you log in as root (or use `su -` from the shell to become root), your `$PATH` variable is set to include some directories that contain commands for the root user. In the past, these have included the following:

`/sbin`: Originally contained commands needed to boot your system, including commands for checking filesystems (`fsck`) and turning on swap devices (`swapon`).

/usr/sbin: Originally contained commands for such things as managing user accounts (such as `useradd`) and checking processes that are holding files open (such as `lsof`). Commands that run as daemon processes are also contained in this directory. *Daemon processes* are processes that run in the background, waiting for service requests such as those to access a printer or a web page. (Look for commands that end in `d`, such as `sshd`, `pppd`, and `cupsd`.)

For the latest Ubuntu, RHEL and Fedora releases, all administrative commands from the two directories are stored in the `/usr/sbin` directory (which is symbolically linked from `/sbin`). Also, only `/usr/sbin` is added to the PATH of the root user, as well as the PATH of all regular users.

Some administrative commands are contained in regular user directories (such as `/bin` and `/usr/bin`). This is especially true of commands that have some options available to everyone. An example is the `/bin/mount` command, which anyone can use to list mounted filesystems but only root can use to mount filesystems. (Some desktops, however, are configured to let regular users use `mount` to mount CDs, DVDs, or other removable media.)

Note

See the section "Mounting Filesystems" in Chapter 12 for instructions on how to mount a filesystem.

To find commands intended primarily for the system administrator, check out the section 8 manual pages (usually in `/usr/share/man/man8`). They contain descriptions and options for most Linux administrative commands. If you want to add commands to your system, consider adding them to directories such as `/usr/local/bin` or `/usr/local/sbin`. Some Linux distributions automatically add those directories to your PATH, usually before your standard `bin` and `sbin` directories. In that way, commands installed to those directories are not only accessible, but can also override commands of the same name in other directories. Some third-party applications that are not included with Linux distributions are sometimes placed in the `/usr/local/bin`, `/opt/bin`, or `/usr/local/sbin` directory.

Administrative configuration files

Configuration files are another mainstay of Linux administration. Almost everything that you set up for your particular computer—user accounts, network addresses, or GUI preferences—results in settings being stored in plain-text files. This has some advantages and some disadvantages.

The advantage of plain-text files is that it's easy to read and change them. Any text editor will do. The downside, however, is that as you edit configuration files, traditionally no error checking is done. You sometimes have to run the program that reads these files (such as a network daemon or the X desktop) to find out whether you set up the files correctly.

While some configuration files use standard structures, such as XML for storing information, many do not. So, you need to learn the specific structure rules for each configuration

file. A comma or a quote in the wrong place can sometimes cause an entire interface to fail. You can check in many ways that the structure of many configuration files is correct.

Some software packages offer a command to test the sanity of the configuration file tied to a package before you start a service. For example, the `testparm` command is used with Samba to check the sanity of your `smb.conf` file. Other times, the daemon process providing a service offers an option for checking your config file. For example, run `httpd -t` to check your Apache web server configuration before starting your web server.

NOTE

Some text editors, such as the `vim` command (not `vi`), understand the structure of some types of configuration files. If you open such a configuration file in `vim`, notice that different elements of the file are shown in different colors. In particular, you can see comment lines in a different color than data.

Throughout this book, you'll find descriptions of the configuration files that you need to set up the different features that make up Linux systems. The two major locations of configuration files are your home directory (where your personal configuration files are kept) and the `/etc` directory (which holds system-wide configuration files).

Following are descriptions of directories (and subdirectories) that contain useful configuration files. The descriptions are followed by some individual configuration files in `/etc` that are of particular interest. Viewing the contents of Linux configuration files can teach you a lot about administering Linux systems.

`$HOME`: All users store in their home directories information that directs how their login accounts behave. Many configuration files are stored directly in each user's home directory (such as `/home/joe`) and begin with a dot (.), so they don't appear in a user's directory when you use a standard `ls` command (you need to type **`ls -a`** to see them). Likewise, dot files and directories won't show up in most file manager windows by default. There are dot files that define the behavior of each user's shell, the desktop look-and-feel, and options used with your text editor. There are even files such as those in each user's `$HOME/.ssh` directory that configure permissions for logging into remote systems. (To see the name of your home directory, type **`echo $HOME`** from a shell.)

`/etc`: This directory contains most of the basic Linux system configuration files.

`/etc/cron*`: Directories in this set contain files that define how the `crond` utility runs applications on a daily (`cron.daily`), hourly (`cron.hourly`), monthly (`cron.monthly`), or weekly (`cron.weekly`) schedule.

`/etc/cups`: Contains files used to configure the CUPS printing service.

`/etc/default`: Contains files that set default values for various utilities. For example, the file for the `useradd` command defines the default group number, home directory, password expiration date, shell, and skeleton directory (`/etc/skel`) used when creating a new user account.

/etc/httpd: Contains a variety of files used to configure the behavior of your Apache web server (specifically, the `httpd` daemon process). (On Ubuntu and other Linux systems, `/etc/apache` or `/etc/apache2` is used instead.)

/etc/mail: Contains files used to configure your `sendmail` mail transport agent.

/etc/postfix: Contains configuration files for the `postfix` mail transport agent.

/etc/ppp: Contains several configuration files used to set up Point-to-Point Protocol (PPP) so that you can have your computer dial out to the Internet. (PPP was more commonly used when dial-up modems were popular.)

/etc/rc?.d: There is a separate `rc?.d` directory for each valid system state: `rc0.d` (shutdown state), `rc1.d` (single-user state), `rc2.d` (multiuser state), `rc3.d` (multiuser plus networking state), `rc4.d` (user-defined state), `rc5.d` (multiuser, networking, plus GUI login state), and `rc6.d` (reboot state). These directories are maintained for compatibility with old UNIX SystemV init services.

/etc/security: Contains files that set a variety of default security conditions for your computer, basically defining how authentication is done. These files are part of the `pam` (pluggable authentication modules) package.

/etc/skel: Any files contained in this directory are automatically copied to a user's home directory when that user is added to the system. By default, most of these files are dot (.) files, such as `.kde` (a directory for setting KDE desktop defaults) and `.bashrc` (for setting default values used with the bash shell).

/etc/sysconfig: Contains important system configuration files that are created and maintained by various services (including `firewalld`, `samba`, and most networking services). These files are critical for Linux distributions, such as Fedora and RHEL, that use GUI administration tools but are not used on other Linux systems at all.

/etc/systemd: Contains files associated with the `systemd` facility, for managing the boot process and system services. In particular, when you run `systemctl` commands to enable and disable services, files that make that happen are stored in subdirectories of the `/etc/systemd system` directory.

/etc/xinetd.d: Contains a set of files, each of which defines an on-demand network service that the `xinetd` daemon listens for on a particular port. When the `xinetd` daemon process receives a request for a service, it uses the information in these files to determine which daemon processes to start to handle the request.

The following are some interesting configuration files in `/etc`:

aliases: Can contain distribution lists used by the Linux mail services. (This file is located in `/etc/mail` in Ubuntu when you install the `sendmail` package.)

bashrc: Sets system-wide defaults for bash shell users. (This may be called `bash.bashrc` on some Linux distributions.)

crontab: Sets times for running automated tasks and variables associated with the `cron` facility (such as the `SHELL` and `PATH` associated with `cron`).

csh.cshrc (or **cshrc**): Sets system-wide defaults for `csh` (C shell) users.

exports: Contains a list of local directories that are available to be shared by remote computers using the Network File System (NFS).

fstab: Identifies the devices for common storage media (hard disk, DVD, CD-ROM, and so on) and locations where they are mounted in the Linux system. This is used by the `mount` command to choose which filesystems to mount when the system first boots.

group: Identifies group names and group IDs (GIDs) that are defined on the system. Group permissions in Linux are defined by the second of three sets of `rwx` (read, write, execute) bits associated with each file and directory.

gshadow: Contains shadow passwords for groups.

host.conf: Used by older applications to set the locations in which domain names (for example, `redhat.com`) are searched for on TCP/IP networks (such as the Internet). By default, the local hosts file is searched and then any name server entries in `resolv.conf`.

hostname: Contains the hostname for the local system (beginning in RHEL 7 and recent Fedora and Ubuntu systems).

hosts: Contains IP addresses and hostnames that you can reach from your computer. (Usually this file is used just to store names of computers on your LAN or small private network.)

inittab: On earlier Linux systems, contained information that defined which programs start and stop when Linux boots, shuts down, or goes into different states in between. This configuration file was the first one read when Linux started the `init` process. This file is no longer used on Linux systems that support `systemd`.

mtab: Contains a list of filesystems that are currently mounted.

mtools.conf: Contains settings used by DOS tools in Linux.

named.conf: Contains DNS settings if you are running your own DNS server (`bind` or `bind9` package).

nsswitch.conf: Contains name service switch settings, for identifying where critical system information (user accounts, hostname-to-address mappings, and so on) comes from (local host or via network services).

ntp.conf: Includes information needed to run the Network Time Protocol (NTP).

passwd: Stores account information for all valid users on the local system. Also includes other information, such as the home directory and default shell. (Rarely includes the user passwords themselves, which are typically stored in the `/etc/shadow` file.)

printcap: Contains definitions for the printers configured for your computer. (If the `printcap` file doesn't exist, look for printer information in the `/etc/cups` directory.)

profile: Sets system-wide environment and startup programs for all users. This file is read when the user logs in.

protocols: Sets protocol numbers and names for a variety of Internet services.

rpc: Defines remote procedure call names and numbers.

services: Defines TCP/IP and UDP service names and their port assignments.

shadow: Contains encrypted passwords for users who are defined in the `passwd` file. (This is viewed as a more secure way to store passwords than the original encrypted password in the `passwd` file. The `passwd` file needs to be publicly readable, whereas the `shadow` file can be unreadable by all but the root user.)

shells: Lists the shell command-line interpreters (`bash`, `sh`, `csh`, and so on) that are available on the system as well as their locations.

sudoers: Sets commands that can be run by users, who may not otherwise have permission to run the command, using the `sudo` command. In particular, this file is used to provide selected users with root permission.

rsyslog.conf: Defines what logging messages are gathered by the `rsyslogd` daemon and in which files they are stored. (Typically, log messages are stored in files contained in the `/var/log` directory.)

xinetd.conf: Contains simple configuration information used by the `xinetd` daemon process. This file mostly points to the `/etc/xinetd.d` directory for information about individual services.

Another directory, `/etc/X11`, includes subdirectories that each contain system-wide configuration files used by X and different X window managers available for Linux. The `xorg.conf` file (configures your computer and monitor to make it usable with X) and configuration directories containing files used by `xdm` and `xinit` to start X are in here.

Directories relating to window managers contain files that include the default values that a user will get if that user starts one of these window managers on your system. The twm window manager may have system-wide configuration files in these directories.

Administrative log files and systemd journal

One of the things that Linux does well is keep track of itself. This is a good thing when you consider how much is going on in a complex operating system.

Sometimes you are trying to get a new facility to work and it fails without giving you the foggiest reason why. Other times, you want to monitor your system to see whether people are trying to access your computer illegally. In any of those cases, you want to be able to refer to messages coming from the kernel and services running on the system.

For Linux systems that don't use the systemd facility, the main utility for logging error and debugging messages is the rsyslogd daemon. (Some older Linux systems use syslogd and syslogd daemons.) Although you can still use rsyslogd with systemd systems, systemd has its own method of gathering and displaying messages called the systemd journal (journalctl command).

Using journalctl to view the systemd journal

The primary command for viewing messages from the systemd journal is the journalctl command. The boot process, the kernel, and all systemd-managed services direct their status and error messages to the systemd journal.

Using the journalctl command, you can display journal messages in many different ways. Here are some examples:

```
# journalctl
# journalctl --list-boots | head
-2 93bdb6164... Sat 2020-01-04 21:07:28 EST—Sat 2020-01-04 21:19:37 EST
-1 7336cb823... Sun 2020-01-05 10:38:27 EST—Mon 2020-01-06 09:29:09 EST
 0 eaebac25f... Sat 2020-01-18 14:11:41 EST—Sat 2020-01-18 16:03:37 EST
# journalctl -b 488e152a3e2b4f6bb86be366c55264e7
# journalctl -k
```

In these examples, the journalctl command with no options lets you page through all messages in the systemd journal. To list the boot IDs for each time the system was booted, use the –list-boots option. To view messages associated with a particular boot instance, use the -b option with one of the boot instances. To see only kernel messages, use the -k option. Here are some more examples:

```
# journalctl _SYSTEMD_UNIT=sshd.service
# journalctl PRIORITY=0
# journalctl -a -f
```

Use the _SYSTEMD_UNIT= options to show messages for specific services (here, the sshd service) or for any other systemd unit file (such as other services or mounts). To see messages associated with a particular syslog log level, set PRIORITY= to a value from 0 to 7. In this case, only emergency (0) messages are shown. To follow messages as they come in, use the -f option; to show all fields, use the -a option.

Managing log messages with rsyslogd

The rsyslogd facility, and its predecessor syslogd, gather log messages and direct them to log files or remote log hosts. Logging is done according to information in the /etc/rsyslog.conf file. Messages are typically directed to log files that are usually in the /

var/log directory, but they can also be directed to log hosts for additional security. Here are a few common log files:

boot.log: Contains boot messages about services as they start up.

messages: Contains many general informational messages about the system.

secure: Contains security-related messages, such as login activity or any other act that authenticates users.

Refer to Chapter 13, "Understanding Server Administration," for information on configuring the rsyslogd facility.

Using Other Administrative Accounts

You don't hear much about logging in with other administrative user accounts (besides root) on Linux systems. It was a fairly common practice in UNIX systems to have several different administrative logins that allowed administrative tasks to be split among several users. For example, people sitting near a printer could have lp permissions to move print jobs to another printer if they knew a printer wasn't working.

In any case, administrative logins are available with Linux; however, logging in directly as those users is disabled by default. The accounts are maintained primarily to provide ownership for files and processes associated with particular services. When daemon processes are run under separate administrative logins, having one of those processes cracked does not give the cracker root permission and the ability to access other processes and files. Consider the following examples:

lp: User owns such things as the /var/log/cups printing log file and various printing cache and spool files. The home directory for lp is /var/spool/lpd.

apache: User can set up content files and directories on an Apache web server. It is primarily used to run the web server processes (httpd) in RHEL and Fedora systems, while the www-data user runs the Apache service (apache2) on Ubuntu systems.

avahi: User runs the avahi daemon process to provide zeroconf services on your network.

chrony: User runs the chronyd daemon, which is used to maintain accurate computer clocks.

postfix: User owns various mail server spool directories and files. The user runs the daemon processes used to provide the postfix service (master).

bin: User owns many commands in /bin in traditional UNIX systems. This is not the case in some Linux systems (such as Ubuntu, Fedora, and Gentoo) because root owns most executable files. The home directory of bin is /bin.

news: User could do administration of Internet news services, depending on how you set permission for /var/spool/news and other news-related resources. The home directory for news is /etc/news.

rpc: User runs the remote procedure calls daemon (rpcbind), which is used to receive calls for services on the host system. The NFS service uses the RPC service.

By default, the administrative logins in the preceding list are disabled. You would need to change the default shell from its current setting (usually /sbin/nologin or /bin/false) to a real shell (typically /bin/bash) to be able to log in as these users. As mentioned earlier, however, they are really not intended for interactive logins.

Checking and Configuring Hardware

In a perfect world, after installing and booting Linux, all of your hardware is detected and available for access. Although Linux systems have become quite good at detecting hardware, sometimes you must take special steps to get your computer hardware working. Also, the growing use of removable USB devices (CDs, DVDs, flash drives, digital cameras, and removable hard drives) has made it important for Linux to do the following:

- Efficiently manage hardware that comes and goes
- Look at the same piece of hardware in different ways. (For example, it should be able to see a printer as a fax machine, scanner, and storage device as well as a printer.)

Linux kernel features added in the past few years have made it possible to change drastically the way that hardware devices are detected and managed. The Udev subsystem dynamically names and creates devices as hardware comes and goes.

If this sounds confusing, don't worry. It's designed to make your life as a Linux user much easier. The result of features built on the kernel is that device handling in Linux has become more automatic and more flexible:

More automatic For most common hardware, when a hardware device is connected or disconnected, it is automatically detected and identified. Interfaces to access the hardware are added so it is accessible to Linux. Then the fact that the hardware is present (or removed) is passed to the user level, where applications listening for hardware changes are ready to mount the hardware and/or launch an application (such as an image viewer or music player).

More flexible If you don't like what happens automatically when a hardware item is connected or disconnected, you can change it. For example, features built into GNOME and KDE desktops let you choose what happens when a music CD or data DVD is inserted, or when a digital camera is connected. If you prefer that a different program be launched to handle it, you can easily make that change.

The following sections cover several issues related to getting your hardware working properly in Linux. First, it describes how to check information about the hardware components of your system. It then covers how to configure Linux to deal with removable media. Finally, it describes how to use tools for manually loading and working with drivers for hardware that is not detected and loaded properly.

Checking your hardware

When your system boots, the kernel detects your hardware and loads drivers that allow Linux to work with that hardware. Because messages about hardware detection scroll quickly off the screen when you boot, to view potential problem messages you have to redisplay those messages after the system comes up.

There are a few ways to view kernel boot messages after Linux comes up. Any user can run the `dmesg` command to see what hardware was detected and which drivers were loaded by the kernel at boot time. As new messages are generated by the kernel, those messages are also made available to the `dmesg` command.

A second way to see boot messages is the `journalctl` command to show the messages associated with a particular boot instance (as shown earlier in this chapter).

Note

After your system is running, many kernel messages are sent to the `/var/log/messages` file. So, for example, if you want to see what happens when you plug in a USB drive, you can type `tail -f /var/log/messages` and watch as devices and mount points are created. Likewise, you could use the `journalctl -f` command to follow messages as they come into the `systemd` journal.

The following is an example of some output from the `dmesg` command that was trimmed down to show some interesting information:

```
$ dmesg | less
[    0.000000] Linux version 5.0.9-301.fc30.x86_64
   (mockbuild@bkernel04.phx2.fedoraproject.org) (gcc version 9.0.1
20190312
   (Red Hat 9.0.1-0.10) (GCC)) #1 SMP Tue Apr 23 23:57:35 UTC 2019
[    0.000000] Command line:
   BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.0.9-301.fc30.x86_64
   root=/dev/mapper/fedora_localhost--live-root ro
   resume=/dev/mapper/fedora_localhost--live-swap
   rd.lvm.lv=fedora_localhost-live/root
   rd.lvm.lv=fedora_localhost-live/swap rhgb quiet
...
             S31B1102 USB DISK         1100 PQ: 0 ANSI: 0 CCS
[79.177466] sd 9:0:0:0: Attached scsi generic sg2 type 0
```

```
[79.177854] sd 9:0:0:0: [sdb]
             8343552 512-byte logical blocks: (4.27 GB/3.97 GiB)
[79.178593] sd 9:0:0:0: [sdb] Write Protect is off
```

From this output, you first see the Linux kernel version, followed by kernel command-line options. The last few lines reflect a 4GB USB drive being plugged into the computer.

If something goes wrong detecting your hardware or loading drivers, you can refer to this information to see the name and model number of hardware that's not working. Then you can search Linux forums or documentation to try to solve the problem. After your system is up and running, some other commands let you look at detailed information about your computer's hardware. The `lspci` command lists PCI buses on your computer and devices connected to them. Here's a snippet of output:

```
$ lspci
00:00.0 Host bridge: Intel Corporation
     5000X Chipset Memory ControllerHub
00:02.0 PCI bridge: Intel Corporation 5000 Series Chipset
     PCI Express x4 Port 2
00:1b.0 Audio device: Intel Corporation 631xESB/632xESB
     High Definition Audio Controller (rev 09)
00:1d.0 USB controller: Intel Corporation 631xESB/632xESB/3100
     Chipset UHCI USBController#1 (rev 09)
07:00.0 VGA compatible controller: nVidia Corporation NV44
0c:02.0 Ethernet controller: Intel Corporation 82541PI
     Gigabit Ethernet Controller (rev 05)
```

The host bridge connects the local bus to the other components on the PCI bridge. I cut down the output to show information about the different devices on the system that handle various features: sound (Audio device), flash drives and other USB devices (USB controller), the video display (VGA compatible controller), and wired network cards (Ethernet controller). If you are having trouble getting any of these devices to work, noting the model names and numbers gives you something to Google.

To get more verbose output from `lspci`, add one or more `-v` options. For example, using `lspci -vvv`, I received information about my Ethernet controller, including latency, capabilities of the controller, and the Linux driver (e1000) being used for the device.

If you are specifically interested in USB devices, try the `lsusb` command. By default, `lsusb` lists information about the computer's USB hubs along with any USB devices connected to the computer's USB ports:

```
$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 413c:2105 Dell Computer Corp.
    Model L100 Keyboard
```

```
Bus 002 Device 004: ID 413c:3012 Dell Computer Corp.
    Optical Wheel Mouse
Bus 001 Device 005: ID 090c:1000 Silicon Motion, Inc. -
    Taiwan 64MB QDI U2 DISK
```

From the preceding output, you can see the model of a keyboard, mouse, and USB flash drive connected to the computer. As with `lspci`, you can add one or more `-v` options to see more details.

To see details about your processor, run the `lscpu` command. That command gives basic information about your computer's processors.

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
...
```

From the sampling of output of `lscpu`, you can see that this is a 64-bit system (x86-64), it can operate in 32-bit or 64-bit modes, and there are four CPUs.

Managing removable hardware

Linux systems such as Red Hat Enterprise Linux, Fedora, and others, which support full GNOME desktop environments, include simple graphical tools for configuring what happens when you attach popular removable devices to the computer. So, with a GNOME desktop running, you simply plug in a USB device or insert a CD or DVD, and a window may pop up to deal with that device.

Although different desktop environments share many of the same underlying mechanisms (in particular, Udev) to detect and name removable hardware, they offer different tools for configuring how they are mounted or used. Udev (using the `udevd` daemon) creates and removes devices (`/dev` directory) as hardware is added and removed from the computer. Settings that are of interest to someone using a desktop Linux system, however, can be configured with easy-to-use desktop tools.

The Nautilus file manager used with the GNOME desktop lets you define what happens when you attach removable devices or insert removable media into the computer from the File Management Preferences window. The descriptions in this section are based on GNOME 3.32 in Fedora 30.

From the GNOME 3.32 desktop, select Activities and type **Removable Media**. Then select the Removable Media Setting entry.

The following settings are available from the Removable Media window. These settings relate to how removable media are handled when they are inserted or plugged in. In most cases, you are prompted about how to handle a medium that is inserted or connected.

CD audio: When an audio CD is inserted, you can choose to be prompted for what to do (default), do nothing, open the contents in a folder window, or select from various audio CD players to be launched to play the content. Rhythmbox (music player), Audio CD Extractor (CD burner), and Brasero (CD burner) are among the choices that you have for handling an inserted audio CD.

DVD video: When a commercial video DVD is inserted, you are prompted for what to do with that DVD. You can change that default to launch Totem (videos), Brasero (DVD burner), or another media player you have installed (such as MPlayer).

Music player: When inserted media contains audio files, you are asked what to do. You can select to have Rhythmbox or some other music player begin playing the files by selecting that player from this box.

Photos: When inserted media (such as a memory card from a digital camera) contains digital images, you are asked what to do with those images. You can select to do nothing, or you can select to have the images opened in the Shotwell image viewer (the default application for viewing images on the GNOME desktop) or another installed photo manager.

Software: When inserted media contains installable software, the Software window opens by default. To change that behavior (to ask what to do, do nothing, or open the media contents in a folder), you can select the box for those choices.

Other Media: Select the Type box under the Other Media heading to select how less commonly used media are handled. For example, you can select what actions are taken to handle audio DVDs or blank Blu-ray discs, CDs, or DVDs. You can select what applications to launch for Blu-ray video disc, ebook readers, and Picture CDs.

Note that the settings described here are in effect only for the user who is currently logged in. If multiple users have login accounts, each can have their own way of handling removable media.

NOTE

The Totem movie player does not play movie DVDs unless you add extra software to decrypt the DVD. You should look into legal issues and other movie player options if you want to play commercial DVD movies from Linux.

The options to connect regular USB flash drives or hard drives are not listed on this window. If you connect one of those drives to your computer, however, devices are automatically created when you plug them in (named `/dev/sda`, `/dev/sdb`, and so on). Any filesystems found on those devices are automatically mounted on `/run/media/`*`username`*, and you are prompted if you want to open a Nautilus window to view files on those devices. This is done automatically, so you don't have to do any special configuration to make this happen.

When you are finished with a USB drive, right-click the device's name in the Nautilus file manager window and select Safely Remove Drive. This action unmounts the drive and

removes the mount point in the `/run/media/username` directory. After that, you can safely unplug the USB drive from your computer.

Working with loadable modules

If you have added hardware to your computer that isn't properly detected, you might need to load a module manually for that hardware. Linux comes with a set of commands for loading, unloading, and getting information about hardware modules.

Kernel modules are installed in `/lib/modules/` subdirectories. The name of each subdirectory is based on the release number of the kernel. For example, if the kernel were `5.3.8-200.fc30.x86_64`, the `/lib/modules/5.3.8-200.fc30.x86_64` directory would contain drivers for that kernel. Modules in those directories can then be loaded and unloaded as they are needed.

Commands for listing, loading, unloading, and getting information about modules are available with Linux. The following sections describe how to use those commands.

Listing loaded modules

To see which modules are currently loaded into the running kernel on your computer, use the `lsmod` command. Consider the following example:

```
# lsmod
Module                  Size  Used by
vfat                   17411  1
fat                    65059  1 vfat
uas                    23208  0
usb_storage            65065  2 uas
fuse                   91446  3
ipt_MASQUERADE         12880  3
xt_CHECKSUM            12549  1
nfsv3                  39043  1
rpcsec_gss_krb5        31477  0
nfsv4                 466956  0
dns_resolver           13096  1 nfsv4
nfs                   233966  3 nfsv3,nfsv4
.
.
.
i2c_algo_bit           13257  1 nouveau
drm_kms_helper         58041  1 nouveau
ttm                    80772  1 nouveau
drm                   291361  7 ttm,drm_kms_helper,nouveau
ata_generic            12923  0
pata_acpi              13053  0
e1000                 137260  0
i2c_core               55486  5 drm,i2c_i801,drm_kms_helper
```

This output shows a variety of modules that have been loaded on a Linux system, including one for a network interface card (e1000).

To find information about any of the loaded modules, use the modinfo command. For example, you can enter the following:

```
# /sbin/modinfo -d e1000
Intel(R) PRO/1000 Network Driver
```

Not all modules have descriptions available and, if nothing is available, no data are returned. In this case, however, the e1000 module is described as an Intel(R) PRO/1000 Network Driver module. You can also use the -a option to see the author of the module or -n to see the object file representing the module. The author information often has the email address of the driver's creator, so you can contact the author if you have problems or questions about it.

Loading modules

You can load any module (as root user) that has been compiled and installed (to a /lib/modules subdirectory) into your running kernel using the modprobe command. A common reason for loading a module is to use a feature temporarily (such as loading a module to support a special filesystem on some removable media you want to access). Another reason to load a module is to identify that module as one that will be used by a particular piece of hardware that could not be autodetected.

Here is an example of the modprobe command being used to load the parport module, which provides the core functions to share parallel ports with multiple devices:

```
# modprobe parport
```

After parport is loaded, you can load the parport_pc module to define the PC-style ports available through the interface. The parport_pc module lets you optionally define the addresses and IRQ numbers associated with each device sharing the parallel port, as in the following example:

```
# modprobe parport_pc io=0x3bc irq=auto
```

In this example, a device is identified as having an address of 0x3bc, and the IRQ for the device is autodetected.

The modprobe command loads modules temporarily—they disappear at the next reboot. To add the module to your system permanently, add the modprobe command line to one of the startup scripts run at boot time.

Removing modules

Use the rmmod command to remove a module from a running kernel. For example, to remove the module parport _ pc from the current kernel, type the following:

```
# rmmod parport_pc
```

If it is not currently busy, the `parport_pc` module is removed from the running kernel. If it is busy, try killing any process that might be using the device. Then run `rmmod` again. Sometimes, the module you are trying to remove depends on other modules that may be loaded. For instance, the `usbcore` module cannot be unloaded because it is a built-in module:

```
# rmmod usbcore
rmmod: ERROR: Module usbcore is builtin.
```

Instead of using `rmmod` to remove modules, you could use the `modprobe -r` command. With `modprobe -r`, instead of just removing the module you request, you can also remove dependent modules that are not being used by other modules.

Summary

Many features of Linux, especially those that can potentially damage the system or impact other users, require that you gain root privilege. This chapter describes different ways of obtaining root privilege: direct login, `su` command, or `sudo` command. It also covers some of the key responsibilities of a system administrator and components (configuration files, browser-based tools, and so on) that are critical to a system administrator's work.

The next chapter describes how to install a Linux system. Approaches to installing Linux that are covered in that chapter include how to install from live media and from installation media.

Exercises

Use these exercises to test your knowledge of system administration and to explore information about your system hardware. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. From a shell as root user (or using `sudo`), enable Cockpit (`cockpit.socket`) using the `systemctl` command.
2. Open your web browser to the Cockpit interface (9090) on your system.
3. Find all files under the `/var/spool` directory that are owned by users other than root and display a long listing of them.
4. Become the root user using the `su -` command. To prove that you have root privilege, create an empty or plain-text file named `/mnt/test.txt`. Exit the shell when you are finished. If you are using Ubuntu, you must set your root password first (`sudo passwd root`).

5. Log in as a regular user and become root using `su -`. Edit the `/etc/sudoers` file to allow your regular user account to have full root privilege via the `sudo` command.
6. As the user to whom you just gave `sudoers` privilege, use the `sudo` command to create a file called `/mnt/test2.txt`. Verify that the file is there and owned by the root user.
7. Run the `journalctl -f` command and plug a USB drive into a USB port on your computer. If it doesn't mount automatically, mount it on `/mnt/test`. In a second terminal, unmount the device and remove it, continuing to watch the output from `journalctl -f`.
8. Run a command to see what USB devices are connected to your computer.
9. Pretend that you added a TV card to your computer, but the module needed to use it (`bttv`) was not properly detected and loaded. Load the `bttv` module yourself, and then look to see that it was loaded. Were other modules loaded with it?
10. Remove the `bttv` module along with any other modules that were loaded with it. List your modules to make sure that this was done.